1 Problemset 1 - Supervised Learning

1.1 Linear Classifiers (logistic regression and GDA)

Part a)

Recall that the average empirical loss function of logistic regression is:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \ln(h_\theta(x^{(i)})) + (1 - y^{(i)}) \ln(1 - h_\theta(x^{(i)}))$$

where $y^{(i)} \in \{0, 1\}$, $h_\theta(x) = g(\theta^T x)$, and $g(z) = \dfrac{1}{1 + e^{-z}}$

Find the Hessian Matrix H of this function, and show that for any vector z, it holds true that:

$$z^T H z \geq 0.$$

Remark:

- The following result means that H is a positive definite matrix. Therefore, J is a convex function and has no local minima other than the global one, hence gradient descent always works well on it.
- Note that log-likelihood function and empirical loss function have different gradients and hessians (only differ the sign).

Solution:
First, observe that:

$$g'(z) = g(z)(1 - g(z))$$

By using the derivative of logarithm, we obtain:

$$\frac{\partial}{\partial \theta_j} \ln(h_\theta(x^{(i)})) = x_j^{(i)}(1 - h_\theta(x^{(i)}))$$
$$\frac{\partial}{\partial \theta_j} \ln(1 - h_\theta(x^{(i)})) = -x_j^{(i)} h_\theta(x^{(i)})$$

Hence for each $j \in \{1, ...n\}$:

$$\begin{aligned}\frac{\partial}{\partial \theta_j} J(\theta) &= -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} x_j^{(i)}(1 - h_\theta(x^{(i)})) - (1 - y^{(i)}) x_j^{(i)} h_\theta(x^{(i)}) \\ &= \frac{1}{m}\sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}\end{aligned}$$

Thus the second partial derivatives can be computed as:

$$\frac{\partial^2}{\partial\theta_j\partial\theta_k}J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\frac{\partial}{\partial\theta_k}h_\theta(x^{(i)})x_j^{(i)} = \frac{1}{m}\sum_{i=1}^{m}h_\theta(x^{(i)})(1-h_\theta(x^{(i)}))x_j^{(i)}x_k^{(i)}$$

Let D_{pred} be a $m \times m$ matrix where each of its entries is equal to $h_\theta(x^{(i)})(1-h_\theta(x^{(i)})), i \in \{1, ..., m\}$. By the calculation above, we obtain the Hessian matrix:

$$H = \frac{1}{m}X^T D_{pred} X$$

For the proof that H is positive definite, we can compute $z^T H z$ as:

$$z^T H z = \frac{1}{m}\sum_{i=1}^{m}h_\theta(x^{(i)})(1-h_\theta(x^{(i)}))\left[(x^{(i)})^T z\right]^2 \geq 0$$

Part b) - Coding Problem

Follow the instructions in src/p01b logreg.py to train a logistic regression classifier using Newton's Method. Starting with $\theta = \vec{0}$, run Newton's Method until the updates to θ are small: Specifically, train until the first iteration k such that $|\theta_k - \theta_{k-1}| < \epsilon$, where $\epsilon = 1 \times 10^{-5}$. Make sure to write your model's predictions to the file specified in the code.

Solution:

```
class LogisticRegression(LinearModel): #maximize log-likelihood
    def fit(self, x, y):
        m,n = x.shape
        self.theta = np.zeros(n)

        while(True):
            old_theta = self.theta
            h_theta = self.predict(x)

            gradient = np.dot(x.T, y-h_theta)
            hessian = -np.dot(x.T*h_theta*(1-h_theta),x)

            self.theta -= np.dot(np.linalg.inv(hessian),
               gradient)

            if np.linalg.norm(self.theta-old_theta, ord=1) <
               self.eps:
                break
    def predict(self, x):
        linear_theta = np.dot(x, self.theta)
        return 1/(1+ np.exp(-linear_theta))
```

Result: After training on each dataset then giving the corresponding predictions, we obtain the following statistics:

- Train on "dataset1" and evaluate on "dataset1 valid": **Accuracy** = 90%
- Train on "dataset2" and evaluate on "dataset2 valid": **Accuracy** = 91%

Graph of each dataset:

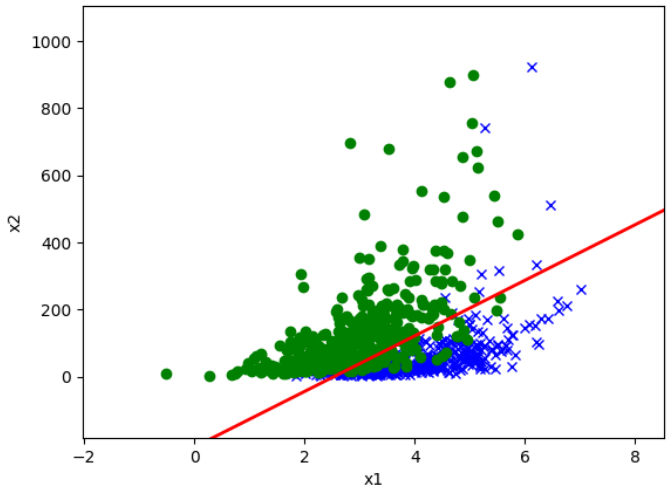

Figure 1: dataset 1 valid

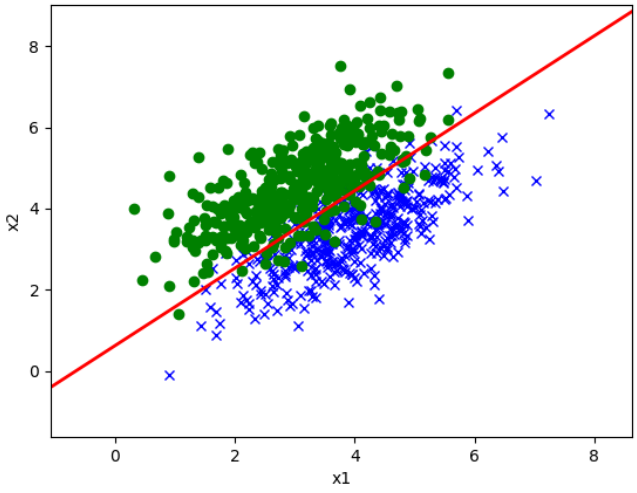

Figure 2: dataset2 valid

Part c)

Recall that in GDA we model the joint distribution of (x; y) by the following equations:

$$\begin{aligned} p(y) &= \phi^y(1-\phi)^{1-y} \\ p(x \mid y=0) &= \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp(-\frac{1}{2}(x-\mu_0)^T\Sigma^{-1}(x-\mu_0)) \\ p(x \mid y=1) &= \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp(-\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1)) \end{aligned}$$

where ϕ, μ_0, μ_1 and Σ are the parameters of our model.
Suppose we have already fit ϕ, μ_0, μ_1 and Σ, and now want to predict y given a new point x. To show that GDA results in a classifier that has a linear decision boundary, show the posterior distribution can be written as:

$$p(y=1 \mid x; parameters) = \frac{1}{1+\exp(-(\theta^T x + \theta_0))}$$

Where $\theta \in \mathbb{R}^n$ and $\theta_0 \in \mathbb{R}$ are appropriate functions of ϕ, μ_0, μ_1 and Σ.

Solution:

For convenience, we may not include the phrase "parameters" after the conditioned x in our probability density function.
By Bayes's theorem, we have:

$$\begin{aligned} p(y=1 \mid x) &= \frac{p(x \mid y=1)p(y=1)}{p(x)} \\ &= \frac{p(x \mid y=1)p(y=1)}{p(y=0)p(x \mid y=0) + p(y=1)p(x \mid y=1)} \\ &= \frac{p(y=1)}{p(y=0)\dfrac{p(x \mid y=0)}{p(x \mid y=1)} + p(y=1)} \\ &= \frac{\phi}{(1-\phi)\exp\left(\dfrac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1) - \dfrac{1}{2}(x-\mu_0)^T\Sigma^{-1}(x-\mu_0)\right) + \phi} \end{aligned}$$

Let $z(\Sigma, \mu_i) = -\frac{1}{2}(x-\mu_i)^T\Sigma^{-1}(x-\mu_i)$. By some computation, we obtain:

$$\begin{aligned} p(y=1 \mid x) &= \frac{1}{1+\exp\left[\left(z(\Sigma,\mu_0) - z(\Sigma,\mu_1) - \ln\left(\frac{1-\phi}{\phi}\right)\right)\right]} \\ &= \frac{1}{1+\exp(-(\theta^T x + \theta_0))} \end{aligned}$$

Whereas:

$$\begin{aligned} \theta &= -\Sigma^{-1}(\mu_0 - \mu_1) \\ \theta_0 &= \frac{1}{2}(\mu_0+\mu_1)^T\Sigma^{-1}(\mu_0-\mu_1) - ln\left(\frac{1-\phi}{\phi}\right) \end{aligned}$$

Part d)

For this part of the problem only, you may assume n (the dimension of x) is 1, so that $\Sigma = [\sigma^2]$ is just a real number, and likewise the determinant of Σ is given by $|\Sigma| = \sigma^2$. Given the dataset, we claim that the maximum likelihood estimates of the parameters are given by:

$$\begin{aligned} \phi &= \frac{1}{m}\sum_{i=1}^{m} 1\{y^{(i)}=1\} \\ \mu_0 &= \frac{\sum_{i=1}^{m} x^{(i)} 1\{y^{(i)}=0\}}{\sum_{i=1}^{m} 1\{y^{(i)}=0\}} \\ \mu_1 &= \frac{\sum_{i=1}^{m} x^{(i)} 1\{y^{(i)}=1\}}{\sum_{i=1}^{m} 1\{y^{(i)}=1\}} \\ \Sigma &= \sum_{i=1}^{m} (x^{(i)} - \mu_{y(i)})(x^{(i)} - \mu_{y^{(i)}})^T \end{aligned}$$

The log-likelihood of the data is:

$$\mathcal{L}(\phi, \mu_0, \mu_1, \Sigma) = \sum_{i=1}^{m} \ln p(x^{(i)}, y^{(i)}; \phi, \mu_0, \mu_1, \Sigma) = \sum_{i=1}^{m} ln\left[p(x^{(i)} \mid y^{(i)}; \mu_0, \mu_1, \Sigma) p(y; \phi)\right]$$

By maximizing $\mathcal{L}$ with respect to the four parameters, prove that the maximum likelihood estimates of ϕ, μ_0, μ_1 and Σ are indeed as given in the formulas above. (You may assume that there is at least one positive and one negative example, so that the denominators in the definitions of μ_0 and μ_1 above are non-zero.)

Solution: Again, we will discard the parameters that come after the conditioned variable for convenience. Since $n = 1$, $x^{(i)} \mid y^{(i)}$ obeys the classic Gaussian distribution:

$$p(x^{(i)} \mid y^{(i)}) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x^{(i)} - \mu_{y^{(i)}})^2}{2\sigma^2}\right)$$

Also:

$$p(y^{(i)}) = \phi^{1\{y^{(i)}=1\}}(1-\phi)^{1\{y^{(i)}=0\}}$$

Thus we can rewrite the log-likelihood function as:

$$\sum_{i=1}^{m} -\frac{(x^{(i)} - \mu_{y^{(i)}})^2}{2\sigma^2} - \ln(\sigma\sqrt{2\pi}) + 1\{y^{(i)} = 1\}\ln(\phi) + 1\{y^{(i)} = 0\}\ln(1-\phi)$$

Computing the partial derivatives:

$$\frac{\partial \phi}{\partial \mathcal{L}} = \sum_{i=1}^{m} \frac{1\{y^{(i)} = 1\}}{\phi} - \frac{1\{y^{(i)} = 0\}}{1-\phi}$$

$$\frac{\partial \mu_{y^{(i)}}}{\partial \mathcal{L}} = -\frac{1}{\sigma^2}\sum_{i=1}^{m}(\mu_{y^{(i)}} - x^{(i)})$$

$$\frac{\partial \sigma}{\partial \mathcal{L}} = \frac{1}{\sigma^3}\sum_{i=1}^{m}(x^{(i)} - \mu_{y^{(i)}})^2 - \frac{1}{\sigma}$$

Letting the partial derivatives equal to zero, we solve for the parameters to obtain

$$\begin{aligned}
\phi &= \frac{1}{m}\sum_{i=1}^{m} 1\{y^{(i)} = 1\} \\
\mu_0 &= \frac{\sum_{i=1}^{m} x^{(i)} 1\{y^{(i)} = 0\}}{\sum_{i=1}^{m} 1\{y^{(i)} = 0\}} \\
\mu_1 &= \frac{\sum_{i=1}^{m} x^{(i)} 1\{y^{(i)} = 1\}}{\sum_{i=1}^{m} 1\{y^{(i)} = 1\}} \\
\sigma^2 &= \sum_{i=1}^{m}(x^{(i)} - \mu_{y^{(i)}})^2
\end{aligned}$$

Part e) - Coding Problem

In src/p01e gda.py, fill in the code to calculate ϕ, μ_0, μ_1 and Σ, use these parameters to derive θ, and use the resulting GDA model to make predictions on the validation set.

Solution:

```python
import numpy as np
import util

from linear_model import LinearModel
from sklearn.metrics import accuracy_score

class GDA(LinearModel):
    #Gaussian Discriminant Analysis.
    def fit(self, x, y):
        # *** START CODE HERE ***
        m,n = x.shape
        self.theta = np.zeros(n+1)

        # Compute phi, mu_0, mu_1, sigma
        y_1 = sum(y == 1)
        phi = y_1 / m
        mu_0 = np.sum(x[y == 0], axis=0) / (m - y_1)
        mu_1 = np.sum(x[y == 1], axis=0) / y_1
        sigma = ((x[y == 0] - mu_0).T.dot(x[y == 0] - mu_0) + (x[y
            == 1] - mu_1).T.dot(x[y == 1] - mu_1)) / m

        # Compute theta
        sigma_inv = np.linalg.inv(sigma)
        self.theta[0] = 0.5 * (mu_0 + mu_1).dot(sigma_inv).dot(mu_0
            - mu_1) - np.log((1 - phi) / phi)
        self.theta[1:] = sigma_inv.dot(mu_1 - mu_0)

        # Return theta
        return self.theta


    def predict(self, x):
        """Make a prediction given new inputs x.
        Args:
            x: Inputs of shape (m, n).
        Returns:
            Outputs of shape (m,).
        """
        # *** START CODE HERE ***
        linear_theta = x.dot(self.theta) #we added intercept to calc
        return 1/(1 + np.exp(-linear_theta))
```

Result: After training on each dataset then giving the corresponding predictions, we obtain the following statistics:

- Train on "dataset1" and evaluate on "dataset1 valid": **Accuracy** = 83%

- Train on "dataset2" and evaluate on "dataset2 valid": **Accuracy** = 91%

Graph of each dataset:

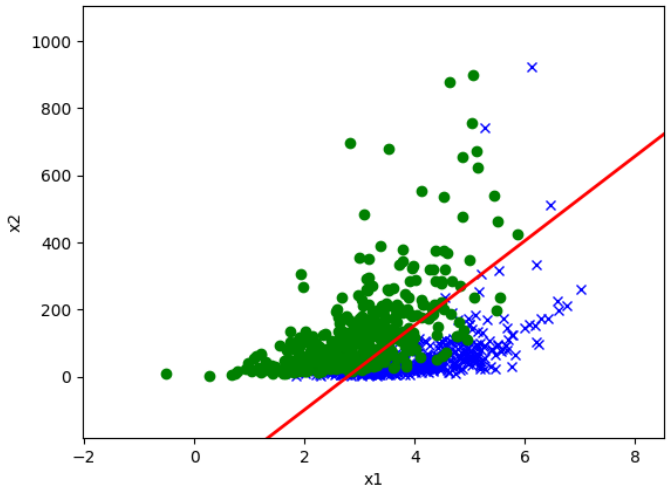

Figure 3: dataset 1 valid

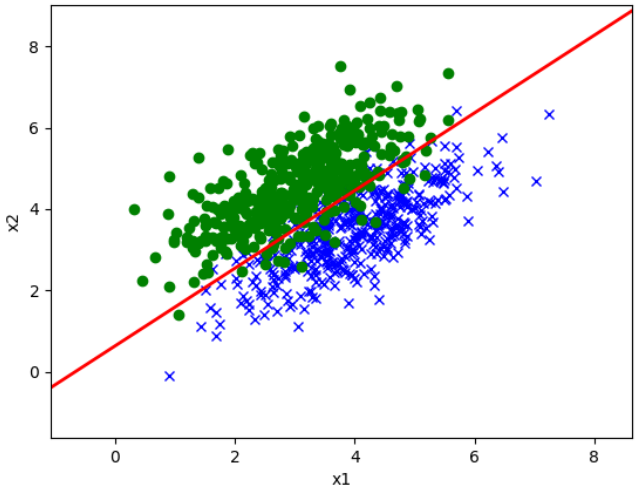

Figure 4: dataset2 valid

Part f) + g)

Plot the results of the two algorithms on each dataset. On which dataset does GDA perform worse than Logistic Regression? Why this might be the case?

Solution: Plotting has been shown above. GDA performs worse in the first dataset, with accuracy of 83% comparing to Logistic Regression's 90%. This is probably due to the fact that the input values x do not distribute Gaussianly when conditioned upon y's.

1.2 Incomplete, Positive-Only Labels

In this problem we will consider training binary classifiers in situations where we do not have full access to the labels. In particular, we consider a scenario, which is not too infrequent in real life, where we have labels only for a subset of the positive examples. All the negative examples and the rest of the positive examples are unlabelled. That is, we assume a dataset $\{(x^{(i)}, t^{(i)}, y^{(i)})\}_{i=1}^{m}$, where $t^{(i)} \in \{0, 1\}$ is the "true" label, and $y^{(i)} = 1$ if $x^{(i)}$ is labeled, zero otherwise.
All labeled examples are positive, which is to say $p(t^{(i)} = 1 \mid y^{(i)} = 1) = 1$, but unlabeled examples may be positive or negative. Our goal in the problem is to construct a binary classifier h of the true label t, with only access to the partial labels y. In other words, we want to construct h such that $h(x^{(i)}) \approx p(t^{(i)} = 1 \mid x^{(i)})$ as closely as possible, using only x and y.
Real world example: Suppose we maintain a database of proteins which are involved in transmitting signals across membranes. Every example added to the database is involved in a signaling process, but there are many proteins involved in cross-membrane signaling which are missing from the database. It would be useful to train a classifier to identify proteins that should be added to the database. In our notation, each example $x^{(i)}$ corresponds to a protein,$y^{(i)} = 1$ if the protein is in the database and 0 otherwise, and $t^{(i)} = 1$ if the protein is involved in a cross-membrane signaling process and thus should be added to the database, and 0 otherwise.

Part a)

Suppose that each $y^{(i)}$ and $x^{(i)}$ are conditionally independent given $t^{(i)}$:

$$p(y^{(i)} = 1 \mid t^{(i)} = 1, x^{(i)}) = p(y^{(i)} = 1, t^{(i)} = 1)$$

Note this is equivalent to saying that labeled examples were selected uniformly at random from the set of positive examples. Prove that the probability of an example being labeled differs by a constant factor from the probability of an example being positive. That is, show that $p(t^{(i)} = 1 \mid x^{(i)}) = \frac{1}{\alpha} p(y^{(i)} = 1 \mid x^{(i)})$, for any value of $x^{(i)}$, where α is a real constant.

Solution: By the rule of conditional probability, we have:

$$p(t^{(i)} = 1 \mid x^{(i)}) = \frac{p(t^{(i)} = 1, x^{(i)})}{p(x^{(i)})}$$
$$p(y^{(i)} = 1 \mid x^{(i)}) = \frac{p(y^{(i)} = 1, x^{(i)})}{p(x^{(i)})}$$

Thus, it suffices to show that the two numerators differ by a real constant.
Again, applying the rule of conditional probability, note that $p(t^{(i)} = 1 \mid y^{(i)} = 1) = 1$:

$$p(y^{(i)} = 1, x^{(i)}) = p(y^{(i)} = 1, x^{(i)}, t^{(i)} = 1) = p(y^{(i)} = 1 \mid t^{(i)} = 1, x^{(i)})p(t^{(i)} = 1, x^{(i)})$$

By using the independence that the problem gave us, we have:

$$p(y^{(i)} = 1, x^{(i)}) = p(y^{(i)} = 1 \mid t^{(i)} = 1)p(t^{(i)} = 1, x^{(i)})$$

Since $p(y^{(i)} = 1 \mid t^{(i)} = 1)$ is independent of $x^{(i)}$, it is a constant. Letting $\alpha = p(y^{(i)} = 1 \mid t^{(i)} = 1)$, we obtain the desired result.

Part b)

Suppose we want to estimate α using a trained classifier h and a held-out validation set V . Let V_+ be the set of labeled (and hence positive) examples in V , given by $V+ = \{x^{(i)} \in V \mid y^{(i)} = 1\}$. Assuming that $h(x^{(i)}) \approx p(y^{(i)} = 1 \mid x^{(i)})$ for all examples $x^{(i)}$, show that:

$$h(x^{(i)}) \approx \alpha, \forall x^{(i)} \in V_+$$

You may assume that $p(t^{(i)} = 1 \mid x^{(i)}) \approx 1$ when $x^{(i)} \in V_+$

Solution: From part a), we have:

$$p(y^{(i)} = 1, x^{(i)}) = p(y^{(i)} = 1 \mid t^{(i)} = 1)p(t^{(i)} = 1, x^{(i)})$$

Thus from $h(x^{(i)}) \approx p(y^{(i)} = 1 \mid x^{(i)})$ and $p(t^{(i)} = 1 \mid x^{(i)}) \approx 1$, we obtain the desired result.

Part c) - Coding Problem

First we will consider the ideal case, where we have access to the true t-labels for training. In **src/p02cde_posonly**, write a logistic regression classifier that uses x_1 and x_2 as input features, and train it using the t-labels (you can ignore the y-labels for this part). Output the trained model's predictions on the test set to the file specified in the code.

Solution:

```
def main(train_path, valid_path, test_path, pred_path):
    #Problem 2: Logistic regression for incomplete, positive-only
        labels.
    pred_path_c = pred_path.replace(WILDCARD, 'c')
    pred_path_d = pred_path.replace(WILDCARD, 'd')
    pred_path_e = pred_path.replace(WILDCARD, 'e')

    # *** START CODE HERE ***


    ###### Prob C
```

```
    x_train, t_train = util.load_dataset(train_path, label_col = 't'
        , add_intercept=True)

    model_t = LogisticRegression(eps = 1e-5)
    model_t.fit(x_train, t_train)

    x_test, t_test = util.load_dataset(test_path, label_col='t',
        add_intercept=True)
    util.plot(x_test, t_test, model_t.theta, 'output/p02c.png')

    t_pred = model_t.predictBin(x_test)
    np.savetxt(pred_path_c, t_pred, fmt='%d')
```

Part d) - Coding Problem

We now consider the case where the t-labels are unavailable, so you only have access to the y-labels at training time. Add to your code in **p02cde_posonly** to re-train the classifier (still using x_1 and x_2 as input features), but using the y-labels only.

Solution:

```
def main(train_path, valid_path, test_path, pred_path):
    pred_path_c = pred_path.replace(WILDCARD, 'c')
    pred_path_d = pred_path.replace(WILDCARD, 'd')
    pred_path_e = pred_path.replace(WILDCARD, 'e')

    # *** START CODE HERE ***

    ###### Prob D

    x_train, y_train = util.load_dataset(train_path, label_col='y',
        add_intercept=True)


    model_y = LogisticRegression()
    model_y.fit(x_train, y_train)

    x_test, y_test = util.load_dataset(test_path, label_col='y',
        add_intercept=True)
    util.plot(x_test, y_test, model_y.theta, 'output/p02d.png')

    y_pred = model_y.predictBin(x_test)
    np.savetxt(pred_path_d, y_pred, fmt='%d')
```

1.3 Poisson Regression

Part a)

Consider the Poisson distribution parameterized by λ :

$$p(y;\lambda) = \frac{e^{-\lambda}\lambda^y}{y!}$$

Show that the Poisson distribution is in the exponential family, and clearly state the values for $b(y), \eta, T(y)$ and $a(\eta)$.

Solution:

We notice that p above is a probability mass function, as Poisson is a discrete random variable. From the problem statement:

$$p(y;\lambda) = \frac{e^{-\lambda}\lambda^y}{y!} = \frac{1}{y!}\exp\left(y\ln(\lambda) - \lambda\right)$$

Hence Poisson distribution belongs to the exponential family, whereas:

$$\begin{aligned} b(y) &= \frac{1}{y!} \\ \eta &= \ln(\lambda) \\ T(y) &= y \\ a(\eta) &= \lambda = e^{\eta} \end{aligned}$$

Part b)

Consider performing regression using a GLM model with a Poisson response variable. What is the canonical response function for the family? (You may use the fact that a Poisson random variable with parameter λ has mean λ.)

Solution:

Canonical response function: $g(\eta) = \mathbb{E}[T(y);\eta] = \lambda = e^{\eta}$

Part c)

For a training set $\{\left(x^{(i)}, y^{(y)}\right); i = 1, ...m\}$, let log-likelihood of an example be $\ln p(y^{(i)} \mid x^{(i)}; \theta)$. By taking the derivative of the log-likelihood with respect to θ_j, derive the stochastic gradient ascent update rule for learning using a GLM model with Poisson responses y and the canonical response function.

Solution:

The log-likelihood with respect to one training example is equal to:

$$\begin{aligned}\mathcal{L}(\theta) = \ln p(y^{(i)} \mid x^{(i)}; \theta) &= -e^{\eta} + \eta \cdot y^{(i)} - \ln(y^{(i)}!) \\ &= -e^{\theta^T x^{(i)}} + \theta^T x^{(i)} \cdot y^{(i)} - \ln(y^{(i)}!)\end{aligned}$$

Computing the partial derivative of $\mathcal{L}$:

$$\frac{\partial}{\partial \theta_j}\mathcal{L}(\theta) = -x_j^{(i)} e^{\theta^T x^{(i)}} + x_j^{(i)} y^{(i)} = (y^{(i)} - e^{\theta^T x^{(i)}}) x_j^{(i)}$$

Hence:

$$\nabla \mathcal{L}(\theta) = (y^{(i)} - e^{\theta^T x^{(i)}}) x^{(i)}$$

Therefore, the gradient ascent learning rule is:

$$\begin{aligned}\theta &:= \theta + \alpha \nabla \mathcal{L}(\theta) \\ &:= \theta + \alpha (y^{(i)} - e^{\theta^T x^{(i)}}) x^{(i)}\end{aligned}$$

where α is the learning rate.

Part d) - Coding Problem

Consider a website that wants to predict its daily traffic. The website owners have collected a dataset of past traffic to their website, along with some features which they think are useful in predicting the number of visitors per day. The dataset is split into train/valid/test sets and follows the same format as Datasets 1-3:
data/ds4_train,valid.csv
We will apply Poisson regression to model the number of visitors per day. Note that applying Poisson regression in particular assumes that the data follows a Poisson distribution whose natural parameter is a linear combination of the input features (i.e., $\eta = \theta^T x$).
In **src/p03d poisson.py**, implement Poisson regression for this dataset and use gradient ascent to maximize the log-likelihood of θ

Solution:

```python
class PoissonRegression(LinearModel):
    def fit(self, x, y):
        m,n = x.shape

        self.theta = np.zeros(n)

        while(True):
            old_theta = self.theta
            h_theta = self.predict(x)

            self.theta = self.theta + self.step_size * np.dot(x.T, (
                y - h_theta))/m

            if np.linalg.norm(self.theta - old_theta, ord=1) < self.
                eps:
                 break

    def predict(self, x):
        return np.exp(np.dot(x, self.theta.T))
```

1.4 Convexity of Generalized Linear Models

In this part, we restrict the exponential family distribution as:

$$p(y;\eta) = b(y)\exp(\eta y - a(\eta))$$

Part a)

Derive an expression for the mean of the distribution. Show that $\mathbb{E}[Y \mid X;\theta]$ be represented as the gradient of the log-partition function a with respect to the natural parameter η.

Solution:

By Leibniz integral rule:

$$\frac{\partial}{\partial\eta}\int_{-\infty}^{+\infty} p(y;\eta)dy = \int_{-\infty}^{+\infty}\frac{\partial}{\partial\eta}p(y;\eta)dy$$

Note that $p(y;\eta)$ is a probability distribution, we have:

$$\begin{aligned}
0 &= \frac{\partial}{\partial \eta}\int_{-\infty}^{+\infty} p(y;\eta)dy \\
&= \int_{-\infty}^{+\infty} \frac{\partial}{\partial \eta} p(y;\eta)dy \\
&= \int_{-\infty}^{+\infty} p(y;\eta)\left(y - \frac{\partial a}{\partial \eta}\right)dy \\
&= \int_{-\infty}^{+\infty} y.p(y;\eta)dy - \frac{\partial a}{\partial \eta}\int_{-\infty}^{+\infty} p(y;\eta)dy \\
&= \mu - \frac{\partial a}{\partial \eta}
\end{aligned}$$

Hence:

$$\mathbb{E}[Y \mid X;\theta] = \frac{\partial a}{\partial \eta}$$

Part b)

Next, derive an expression for the variance of the distribution. In particular, show that $Var(Y \mid X;\theta)$ can be expressed as the derivative of the mean w.r.t η (i.e., the second derivative of the log-partition function $a(\eta)$ w.r.t the natural parameter η.)

Solution:

Using result from part a), we have:

$$\begin{aligned}
\frac{\partial^2 a}{\partial^2 \eta} &= \frac{\partial}{\partial \eta}\int_{-\infty}^{+\infty} y.p(y;\eta)dy \\
&= \int_{-\infty}^{+\infty} \frac{\partial}{\partial \eta}\Big(y.p(y;\eta)\Big)dy \\
&= \int_{-\infty}^{+\infty} y.p(y;\eta).(y - \frac{\partial a}{\partial \eta})dy \\
&= \int_{-\infty}^{+\infty} y^2 p(y;\eta) - \frac{\partial a}{\partial \eta}\int_{-\infty}^{+\infty} y.p(y;\eta)dy \\
&= \mathbb{E}[y^2;\eta] - \big(\mathbb{E}[y;\eta]\big)^2 \\
&= Var(y;\eta)
\end{aligned}$$

Hence:

$$Var(Y \mid X;\theta) = \frac{\partial^2 a}{\partial^2 \eta}$$

Part c)

Finally, write out the loss function $\mathcal{L}(\theta)$, the negative log-likelihood of the distribution, as a function of θ. Then, calculate the Hessian of the loss w.r.t θ, and show that it is always positive semi-definite. This concludes the proof that negative log-likelihood loss of GLM is convex.

Solution:

The distribution of y given x and parameterized by θ:

$$p(y \mid x; \theta) = b(y) \exp(\theta^T xy - a(\theta^T x))$$

Therefore the negative log-likelihood function is:

$$\mathcal{L}(\theta) = -\ln(p(y \mid x; \theta)) = -\ln(b(y)) - \theta^T xy + a(\theta^T x)$$

Differentiating $\mathcal{L}$:

$$\begin{aligned}
\frac{\partial}{\partial \theta_i}\mathcal{L}(\theta) &= -x_i y + \frac{\partial}{\partial \theta_i} a(\theta^T x) \\
&= -x_i y + \frac{\partial a}{\partial(\theta^T x)} \frac{\partial(\theta^T x)}{\partial \theta_i} \\
&= -x_i y + \frac{\partial a}{\partial(\theta^T x)} x_i \\
\frac{\partial^2}{\partial \theta_i \partial \theta_j}\mathcal{L}(\theta) &= x_i \frac{\partial}{\partial \theta_j}\left(\frac{\partial a}{\partial \theta^T x}\right) \\
&= x_i \frac{\partial \frac{\partial a}{\partial \theta^T x}}{\partial \theta^T x} \cdot \frac{\partial \theta^T x}{\partial \theta_j} \\
&= \sigma^2 x_i x_j
\end{aligned}$$

(Since $\sigma^2 = \dfrac{\partial^2 a}{\partial^2 \eta}$, where σ^2 is the variance)

Hence, the Hessian Matrix of $\mathcal{L}$ is:

$$\nabla^2 \mathcal{L} = \sigma^2 x x^T$$

For any $z \in R^n$:

$$z^T(\nabla^2 \mathcal{L})z = (\sigma z^T x)^2 \geq 0$$

Thus, the negative log-likelihood loss of GLM is convex.

1.5 Locally weighted linear regression

Consider a linear regression problem in which we want to "weight" different training examples differently. Specifically, suppose we want to minimize:

$$J(\theta) = \frac{1}{2} \sum_{i=1}^{m} w^{(i)} \left(\theta^T x^{(i)} - y^{(i)}\right)^2$$

Part a.1)

Show that $J(\theta)$ can also be written as

$$J(\theta) = (X\theta - y)^T W (X\theta - y)$$

for some appropriate matrix W. Specify the matrix W.

Solution:

Simply choose

$$W = \begin{bmatrix} \frac{1}{2}w^{(1)} & 0 & \cdots & 0 \\ 0 & \frac{1}{2}w^{(2)} & \cdots & 0 \\ & & \vdots & \\ 0 & 0 & \cdots & \frac{1}{2}w^{(m)} \end{bmatrix} \in R^{m \times m}$$

Then the equation is trivial due to matrix multiplication.

Part a.2)

If all the $w^{(i)}$'s equal 1, then we saw in class that the normal equation is $X^T X\theta = X^T y$ and that the value of θ that minimizes $J(\theta)$ is given by:

$$(X^T X)^{-1} X^T y$$

By finding the derivative $\nabla_\theta J(\theta)$ and setting that to zero, generalize the normal equation to this weighted setting, and give the new value of θ that minimizes $J(\theta)$ in closed form as a function of X, W and y.

Solution:

Applying the chain rule:

$$\begin{aligned} \nabla_\theta J(\theta) &= \nabla_\theta (X\theta - y)^T W (X\theta - y) \\ &= \nabla_\theta \big[(X\theta - y)\big] \nabla_{X\theta - y} \big[(X\theta - y)^T W (X\theta - y)\big] \\ &= X^T 2W(X\theta - y) \\ &= 2X^T W X\theta - 2X^T W y \end{aligned}$$

Hence, the optimal value of θ is:

$$\theta = (X^T W X)^{-1} X^T W y$$

Part a.3)

Suppose we have a dataset $\{\left(x^{(i)}; y^{(i)}\right); i = 1, ...m\}$ of m independent examples, but we model the $y^{(i)}$'s as drawn from conditional distributions with different levels of variance $\left(\sigma^{(i)}\right)^2$. Specifically, assume the model:

$$p(y^{(i)} \mid x^{(i)}; \theta) = \frac{1}{\left(\sigma^{(i)}\right)^2\sqrt{2\pi}} \exp\left(\frac{\left(y^{(i)} - \theta^T x^{(i)}\right)^2}{2\left(\sigma^{(i)}\right)^2}\right)$$

That is, each $y^{(i)}$ is drawn from a Gaussian distribution with mean $\theta^T x^{(i)}$ and variance $\left(\sigma^{(i)}\right)^2$ (where the $\sigma^{(i)}$'s are fixed, known, constants). Show that finding the maximum likelihood estimate of θ reduces to solving a weighted linear regression problem. State clearly what the $w^{(i)}$'s are in terms of the $\sigma^{(i)}$'s.

Solution:

Computing the partial derivatives of the log-likelihood function:

$$\frac{\partial J(\theta)}{\partial \theta_k} = -\sum_{i=1}^{m} \frac{\theta^T x^{(i)} - y^{(i)}}{\left(\sigma^{(i)}\right)^2} x_k^{(i)}$$

Hence the gradient vector can be expressed as:

$$\nabla_\theta J(\theta) = \begin{bmatrix} \frac{x_1^{(1)}}{\left(\sigma^{(1)}\right)^2} & \frac{x_1^{(2)}}{\left(\sigma^{(2)}\right)^2} & \cdots & \frac{x_1^{(m)}}{\left(\sigma^{(m)}\right)^2} \\ \frac{x_2^{(1)}}{\left(\sigma^{(1)}\right)^2} & \frac{x_2^{(2)}}{\left(\sigma^{(2)}\right)^2} & \cdots & \frac{x_2^{(m)}}{\left(\sigma^{(m)}\right)^2} \\ & & \vdots & \\ \frac{x_n^{(1)}}{\left(\sigma^{(1)}\right)^2} & \frac{x_n^{(2)}}{\left(\sigma^{(2)}\right)^2} & \cdots & \frac{x_n^{(m)}}{\left(\sigma^{(m)}\right)^2} \end{bmatrix} \begin{bmatrix} \theta^T x^{(1)} - y^{(1)} \\ \theta^T x^{(2)} - y^{(2)} \\ \vdots \\ \theta^T x^{(m)} - y^{(m)} \end{bmatrix}$$

Or:

$$\nabla_\theta J(\theta) = \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & \cdots & x_1^{(m)} \\ x_2^{(1)} & x_2^{(2)} & \cdots & x_2^{(m)} \\ & & \vdots & \\ x_n^{(1)} & x_n^{(2)} & \cdots & x_n^{(m)} \end{bmatrix} \begin{bmatrix} \frac{1}{\left(\sigma^{(1)}\right)^2} & 0 & \cdots & 0 \\ 0 & \frac{1}{\left(\sigma^{(2)}\right)^2} & \cdots & 0 \\ & & \vdots & \\ 0 & 0 & \cdots & \frac{1}{\left(\sigma^{(m)}\right)^2} \end{bmatrix} \begin{bmatrix} \theta^T x^{(1)} - y^{(1)} \\ \theta^T x^{(2)} - y^{(2)} \\ \vdots \\ \theta^T x^{(m)} - y^{(m)} \end{bmatrix}$$

$$= X^T W (X\theta - y)$$

Thus, MLE of θ is equivalent to weighted linear regression problem, whereas $w^{(i)} = \frac{2}{\left(\sigma^{(i)}\right)^2}$

Part b) - Coding Problem

Implement locally weighted linear regression using the normal equations you derived in Part (a) and using

$$w^{(i)} = \exp\Big(-\frac{||x^{(i)} - x||_2^2}{2\tau^2}\Big)$$

Train your model on the **train split** using $\tau = 0.5$, then run your model on the **valid** split and report the mean squared error (MSE). Finally plot your model's predictions on the validation set (plot the training set with blue 'x' markers and the validation set with a red 'o' markers). Does the model seem to be under- or overfitting?

Solution:

```
class LocallyWeightedLinearRegression(LinearModel):
    def __init__(self, tau):
        super(LocallyWeightedLinearRegression, self).__init__()
        self.tau = tau
        self.x = None
        self.y = None

    def fit(self, x, y):
        self.x = x
        self.y = y

    def predict(self, x):
        """Make predictions given inputs x.

        Args:
            x: Inputs of shape (m, n).

        Returns:
            Outputs of shape (m,).
        """
        # *** START CODE HERE ***
        m, n = x.shape
        y_pred = np.zeros(m)

        for i in range(m):
            W = np.diag(np.exp(-np.sum((self.x - x[i])**2, axis=1) /
                (2 * self.tau**2)))
            y_pred[i] = np.linalg.inv(self.x.T.dot(W).dot(self.x)).
               dot(self.x.T).dot(W).dot(self.y).T.dot(x[i])

        return y_pred
```

Note: The green circles are valid y's with true labels plotted again their corresponding x's. They are used to be compared with the red circles (predictions on valid set).

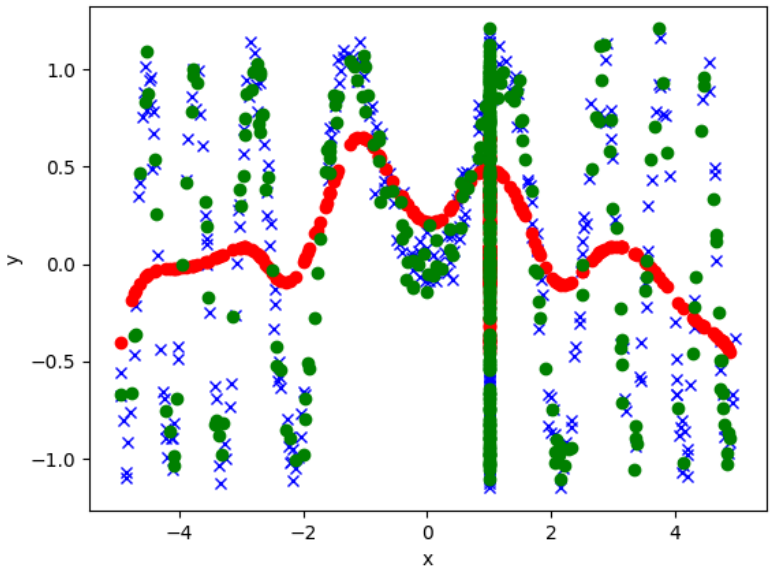

Figure 5: $\tau = 0.5$ - Underfitting, $MSE = 0.33$

Part c) - Coding Problem

We will now tune the hyperparameter τ. Find the MSE value of your model on the validation set for each of the values of τ specified in the code. For each τ, plot your model's predictions on the validation set in the format described in part (b). Report the value of τ which achieves the lowest MSE on the **valid** split, and finally report the MSE on the **test** split using this τ-value.

Solution:

```
def main(tau_values, train_path, valid_path, test_path, pred_path):
    """Problem 5(b): Tune the bandwidth paramater tau for LWR.

    Args:
        tau_values: List of tau values to try.
        train_path: Path to CSV file containing training set.
        valid_path: Path to CSV file containing validation set.
        test_path: Path to CSV file containing test set.
        pred_path: Path to save predictions.
    """
    # Load training set
    x_train, y_train = util.load_dataset(train_path, add_intercept=
        True)
```

```
13
14      # *** START CODE HERE ***
15
16      x_eval, y_eval = util.load_dataset(valid_path, add_intercept=
            True)
17      x_test, y_test = util.load_dataset(test_path, add_intercept=True
            )
18
19      model = LocallyWeightedLinearRegression(tau=0.5)
20      model.fit(x_train, y_train)
21
22      mse_list = []
23      for tau in tau_values:
24          model.tau = tau
25          y_pred = model.predict(x_eval)
26
27          mse = np.mean((y_pred - y_eval)**2)
28          mse_list.append(mse)
29          print(f'valid set: tau={tau}, MSE={mse}')
30
31          plt.figure()
32          plt.title('tau = {}'.format(tau))
33          plt.plot(x_train, y_train, 'bx', linewidth=2)
34          plt.plot(x_eval, y_pred, 'ro', linewidth=2)
35          plt.plot(x_eval, y_eval, 'go', linewidth = 2)
36          plt.xlabel('x')
37          plt.ylabel('y')
38          plt.savefig('output/p05c_tau_{}.png'.format(tau))
39
40      tau_opt = tau_values[np.argmin(mse_list)]
41      print(f'valid set: lowest MSE={min(mse_list)}, tau={tau_opt}')
42      model.tau = tau_opt
43
44      y_pred = model.predict(x_test)
45      np.savetxt(pred_path, y_pred)
46
47      mse = np.mean((y_pred - y_test)**2)
48      print(f'test set: tau={tau_opt}, MSE={mse}')
49      # *** END CODE HERE ***
```

- **Lowest MSE: 0.012** on the valid set where $\tau = 0.05$.
- **With the same hyperparameter**, $MSE = 0.017$ on the test set.

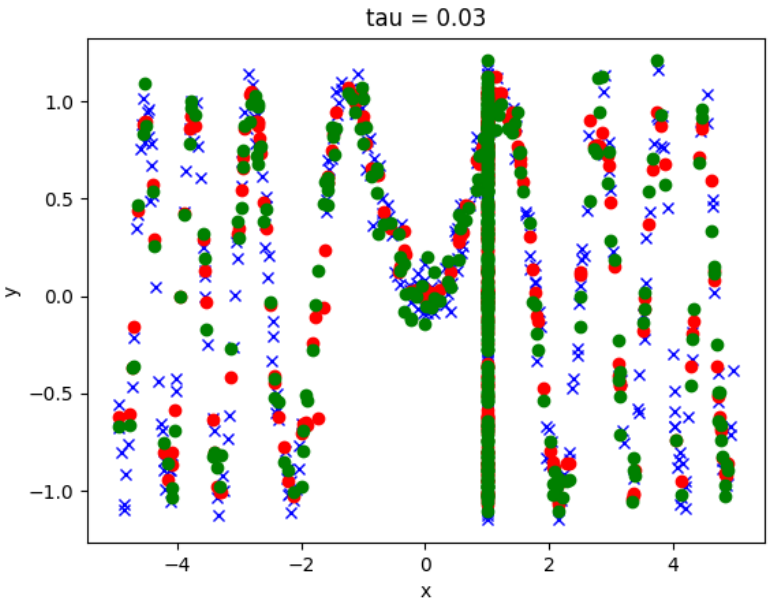

Figure 6: $\tau = 0.03$ - $MSE = 0.018$

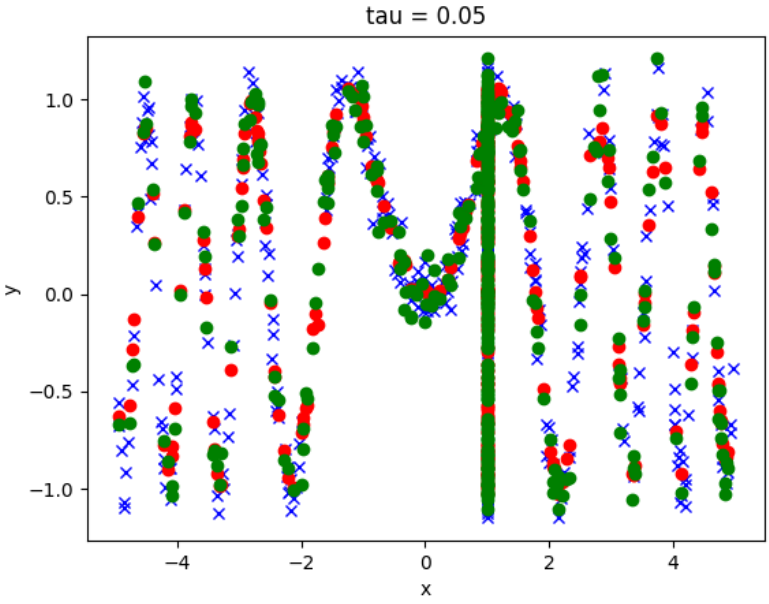

Figure 7: $\tau = 0.05$ - $MSE = 0.012$

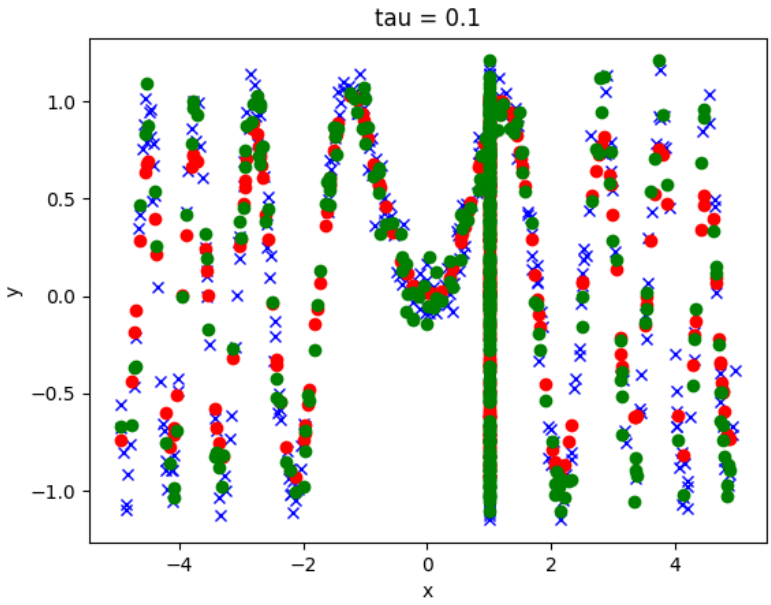

Figure 8: $\tau = 0.1$ - $MSE = 0.024$

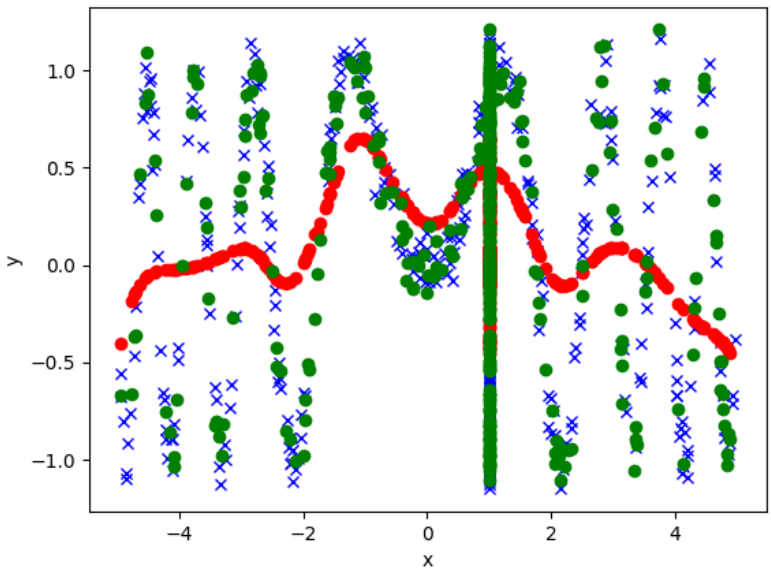

Figure 9: $\tau = 0.5$ - $MSE = 0.33$

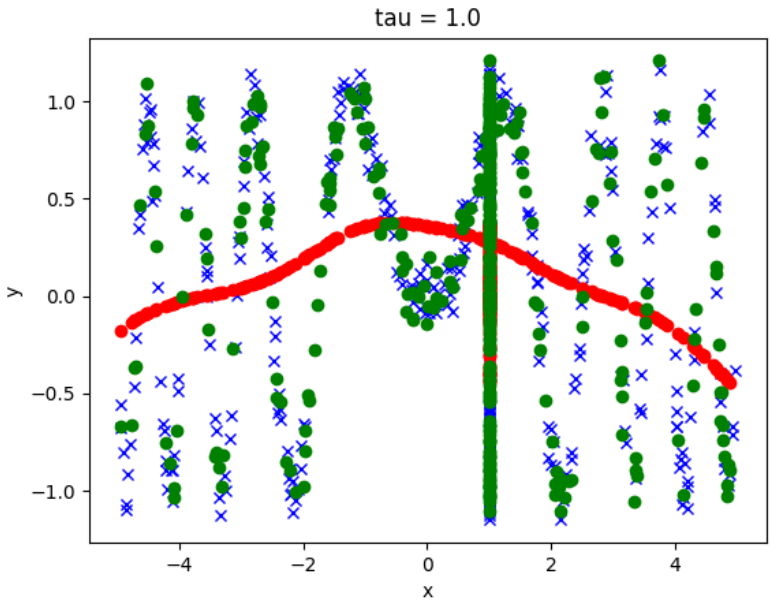

Figure 10: $\tau = 1.0$ - $MSE = 0.4$

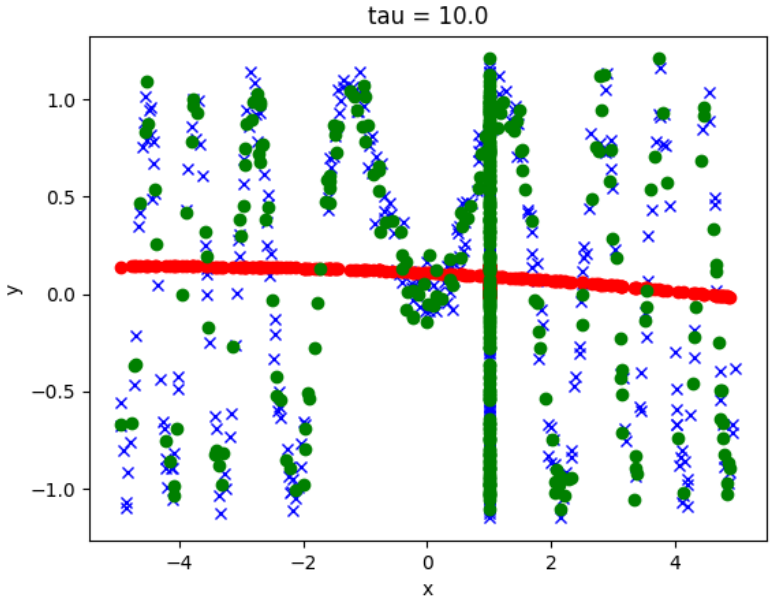

Figure 11: $\tau = 10.0$ - $MSE = 0.43$